Edinburgh Napier University

SET08101 Web Tech

Lab 4 - JavaScript

Dr Simon Wells

1 Aims

At the end of the practical portion of this topic you will:

- Use inline JavaScript
- Use <script> tags
- Use external JavaScript files
- Update HTML & CSS via the DOM

NOTICE: Just as with our other core web languages, HTML & CSS, there is a lot of useful material online. In addition to reading the relevent chapters of the module texts, you should also avail yourself of the following which document JavaScript:

- `https://developer.mozilla.org/bm/docs/Web/JavaScript`
- `https://www.w3schools.com/jsref/default.asp`

2 Activities

We're going to tackle JavaScript a little like we tackled CSS, by examing how JavaScript can interact with HTML.

3 Inline JavaScript

In rare circumstances we might want to add some JavaScript direct to an HTML element like so:

```
<!DOCTYPE html>
<html>
    <head>
        <title>SET08101 - Inline JavaScript</title>
    </head>
    <body>
        <p><a href="#" onClick="alert('Hello World');">Click Me</a></p>
    </body>
</html>
```

Congratulations, you just wrote your first JavaScript[1]. We won't often do this, just like we won't often attach CSS styles direct to an element within HTML. However it does keep things incredibly simple.

Load up the HTML file that you just wrote and try it out. Now let's modify that a little:

```
<!DOCTYPE html>
<html>
    <head>
        <title>SET08101 - Inline JavaScript</title>
    </head>
    <body>
        <p><a href="#" onClick="console.log('Hello World')">Click Me</a></p>
    </body>
</html>
```

[1]Unless you've written some JavaScript before, in which case, well done, you just wrote some more JavaScript.

This time you'll notice that there was no longer any alert box displayed when we clicked the link. Instead, to see where our output went, we need to find our *console*. The console is provided by your browser as part of the Developer Tools[2]. The console is a bit like a command line built into your web browser that will display the output of calls to the console.log() function. This is a really useful function because we'll use it a lot to get output from our code, for example, when we want to test that the value actually stored in a variable actually matches what we think it is[3].

Something else to note in our two example so far is the use of *quoting*; that is the use of the ' and " marks around things. Note that double quotes were used to surround everything associated with onClick= and that within the onClick single quotes were used around the string 'hello world'. There is no rule that says we have to use single quotes inside double quotes, we could have had things the other way around, we just have to be consistent, i.e. ensure that pairs of single quotes match each other and similar for double quotes. We also need to remember that we can't have a pair of double quotes within another set of double quotes, as the opening inner double quote will close the outer double quote and the rest of the line will be, at least, a syntax error. Try it out a bit so you get the idea, e.g.

- onClick='console.log("Hello World")'
- onClick='console.log('Hello World')'
- onClick="console.log("Hello World")"

For each one, see what happens in the console. Get used to checking the output in the console. It's not just for the expected output from calls to console.log() but also will tell you when there are errors.

There are other HTML events, besides onClick, that we can respond to, for example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>SET08101 - Inline JavaScript</title>
    </head>
    <body>
        <p><a href="#" onMouseOver="alert('Hello');">Hover over me</a></p>
    </body>
</html>
```

Note the difference between user interaction, i.e. when the event triggers based on your moving your pointer.

```
<!DOCTYPE html>
<html>
    <head>
        <title>SET08101 - Inline JavaScript</title>
    </head>
    <body>
        <p><a href="#" onMouseOut="alert('Hello');">Hover over me</a></p>
    </body>
</html>
```

Different events give use lots of control over how the user can interact with our pages and the elements that make up our pages. Therea are many other HTML DOM events that your code can respond to. Explore them here: `https://www.w3schools.com/jsref/dom_obj_event.asp`.

3.1 Using Script Blocks

As for CSS, using inline JavaScript feels a bit hacky, good enough for trying something out but not so good if we want to build a larger or more manageable site. Remember, the more mixed up and disorganised things are, the less easily they can be found when needed, or altered with good knowledge of any possible side effects.

[2]If you can remember right back to the week one, this was one of the first things that we got to grips with.
[3]There are alternatives to outputting the values of variables when we are developing, but this is a useful place to start.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
        function helloFunction() { alert('Hello Napier'); }
        </script>
        <title>SET08101 - External JavaScript</title>
    </head>
    <body >
        <p><a href="#" onClick="helloFunction();">Click Me</a></p>
    </body>
</html>
```

3.2 External JavaScript Files

Just as with CSS, we can also put all of our JavaScript into external .js files which can be linked to any html that needs access to the JavaScript code contained therein. This way we can update our JavaScript in just one place, and have our changes available everywhere they are needed. An additional advantage is that external JavaScript files can be cached, this means it is basically saved for reuse. If two pages use the same JavaScript external file then instead of loading it twice, as would happen if the JavaScript was between <script> tags in the respective HTML files, instead it is downloaded for the first page that your user navigates to then is saved in the browser and reused, but not re-downloaded, when referenced from the other file. This can lead to large saving in network bandwidth but also better performance of your sites as loading from local memory is nearly always much faster than reloading across the network.

Let's look at an example. Create a text file called external.html and place the following code into it:

```
<!DOCTYPE html>
<html>
    <head>
         <script src="hello.js"></script>
        <title>SET08101 - External JavaScript</title>
    </head>
    <body >
        <p><a href="#" onClick="helloFunction();">Click Me</a></p>
    </body>
</html>
```

Now create a second text file in the same folder as external.html called hello.js and place the folowing code in it:

```
function helloFunction(){
    alert('Hello Napier');
}
```

As with CSS files, we can place our JavaScript files into their own folders but we must then give the *path* to our JavaScript within the script tag.

Note that you will see <script> tags placed in various places within an HTML file, for example, within the <head> or within the <body> but either is valid.

From this point on you should decide whether to use script blocks or external JavaScript files. It can be easily when trying things out to do everything on the same page, but as soon as you start working on a larger project it can be worth extracting everything out into it's own file.

3.3 Interacting with the DOM

One useful thing to do from JavaScript is to update our HTML from JavaScript code, or *programatically*. Let's try that out:

```
<!DOCTYPE html>
<html>
    <head>
        <title>SET08101 - Interacting with the DOM</title>
    </head>
    <body >
        <p><a href="#" onClick="addMessage();">Click Me</a></p>
        <h1>OUTPUT:</h1>
        <p id="outputDemo"></p>

        <script>
            function addMessage()
            {
                document.getElementById("outputDemo").innerHTML = "HELLO WORLD"
            }
        </script>
    </body>
</html>
```

Rather than a specific HTML typographical element like <p> we can add new content to a more general element like a <div>, e.g.

```
<!DOCTYPE html>
<html>
    <head>
        <title>SET08101 - Interacting with the DOM</title>
    </head>
    <body >
        <p><a href="#" onClick="addMessage();">Click Me</a></p>
        <h1>OUTPUT:</h1>
        <div id="output"></div>

        <script>
            function addMessage()
            {
                document.getElementById("output").innerHTML = "<p>HELLO WORLD</p>"
            }
        </script>
    </body>
</html>
```

It gets even more interesting than this. If we use += instead of just the "=" when assigning our new HTML then we can add additional lines rather than setting things just once.

```
<!DOCTYPE html>
<html>
    <head>
        <title>SET08101 - Interacting with the DOM</title>
    </head>
    <body >
        <p><a href="#" onClick="addMessage();">Click Me</a></p>
        <h1>OUTPUT:</h1>
        <div id="output"></div>

        <script>
            function addMessage()
            {
                document.getElementById("output").innerHTML += "<p>HELLO WORLD</p>"
            }
        </script>
    </body>
</html>
```

Try this out and experiment a little with differen tags. Try setting things up so that each time you click the link a new element will be added to an ordered list.

We can also mix HTML, CSS, and JavaScript, for example:

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <style type="text/css">
            p {color:red}
        </style>
        <title>SET08101 - Interacting with the DOM</title>
    </head>
    <body >
        <p><a href="#" onClick="addMessage();">Click Me</a></p>
        <h1>OUTPUT:</h1>
        <p id="redPara">I am RED</p>

        <script>
            function addMessage()
            {
                document.getElementById("redPara").style.color = "blue";
            }
        </script>
    </body>
</html>
```

What about toggling the colour of our paragraph each time the link is clicked. To do this we'll need a variable to store the current colour in. We'll then need to check the current state and switch colours to the other on each click, for example:

```
<!DOCTYPE html>
<html>
    <head>
        <style type="text/css">
            p {color:red}
        </style>
        <title>SET08101 - Interacting with the DOM</title>
    </head>
    <body >
        <p><a href="#" onClick="colourChanger();">Click Me</a></p>
        <h1>OUTPUT:</h1>
        <p id="redPara">I am RED</p>

        <script>
            var current="red";

            function colourChanger()
            {
                if(current === "red")
                    current = "blue";
                else
                    current = "red";

                document.getElementById("redPara").style.color = current;
            }
        </script>
    </body>
</html>
```

Notice how we've stored the current colour in a variable called current. Also notice that this variable is within the script tags but not within the colourChanger() function. This is because we want our variable to be initialised once when the script is loaded and then updated each time the function is called.

See if you can come up with a way to change the colour of your paragraph and also update the message to match the colour. Consider how you might switch between three, or even four, colours instead of just two.

3.4 Finally

If you want additional practise then the W3Schools JavaScript exercises are a great place to start:

- JavsScript Exercises: `https://www.w3schools.com/js/`

A good way to practice a new language or to improve your existing abilities, not just in JavaScript, but in any language you might be trying to learn, is to try to solve a set of problems using the language. I often use Project Euler when starting out with a new language but there are also many others:

1. Project Euler: `https://projecteuler.net/`
2. Stack Exchange Code Golf: `http://codegolf.stackexchange.com/`
3. Code kata: `http://codekata.com/`
4. Reddit Daily Programmer: `https://www.reddit.com/r/dailyprogrammer`
5. Programming Praxis: `http://programmingpraxis.com/`
6. Rosetta Code: `http://rosettacode.org/wiki/Main_Page`
7. International Collegiate Programming Contest Problems Index: `http://acm.hit.edu.cn/judge/ProblemIndex.php`
8. Algorithmist: `http://www.algorithmist.com/index.php/Main_Page`

Another trick I have is to have a short list of small coding projects that I redo whenever I learn a new language. My personal favourites are to write programs that deal with the following topics (but you should put together your own list if my interests don't match yours):

1. Writing 1D and 2D Cellular Automata
2. Conway's Game of Life
3. Simple text based games, e.g. story-telling or adventure games
4. Quines
5. Simple codes & ciphers

By repeating exercises that you're already familiar with in a new language you can quickly get a feel for a new language does things differently or offers features that make a given problem easier or more difficult to tackle.

This isn't the end of our JavaScript journey, but just the beginning, we're going to spend time getting used to client side JavaScript, then we'll look at server-side JavaScript later on.